How to transfer data to an Excel workbook by using Visual Basic .NET

This article was previously published under Q306022

Article ID : 306022
Last Review : January 6, 2006
Revision : 6.0

For a Microsoft Visual C# .NET version of this article, see 306023 (http://support.microsoft.com/kb/306023/).

On This Page

SUMMARY

This step-by-step article describes several methods for transferring data to Excel 2002 from a Visual Basic .NET program. This article also presents the advantages and disadvantages of each method so that you can select the solution that works best for your situation.

Overview

The technique that is used most frequently to transfer data to an Excel workbook is *Automation*. With Automation, you can call methods and properties that are specific to Excel tasks. Automation gives you the greatest flexibility for specifying the location of your data in the workbook, and the ability to format the workbook and make various settings at run time.

With Automation, you can use several techniques to transfer your data:

- Transfer data cell by cell.
- Transfer data in an array to a range of cells.
- Transfer data in an ADO recordset to a range of cells by using the **CopyFromRecordset** method.
- Create a **QueryTable** object on an Excel worksheet that contains the result of a query on an ODBC or OLEDB data source.
- Transfer data to the clipboard, and then paste the clipboard contents into an Excel worksheet.

You can also use several methods that do not necessarily require Automation to transfer data to Excel. If you are running a server-side program, this can be a good approach for taking the bulk of data processing away from your clients.

The following approaches may be used to transfer your data without Automation:

- Transfer your data to a tab- or comma-delimited text file that Excel can later parse into cells on a worksheet.
- Transfer your data to a worksheet using ADO.NET.
- Transfer XML data to Excel (version 2002 only) to provide data that is formatted and arranged into rows and columns.

Techniques

Use Automation to transfer data cell by cell

With Automation, you can transfer data to a worksheet one cell at a time, as follows.

```
Dim oExcel As Object
Dim oBook As Object
Dim oSheet As Object
```



```
oSheet.Range("A1").Value = "Order ID"
oSheet.Range("B1").Value = "Amount"
oSheet.Range("C1").Value = "Tax"

'Transfer the array to the worksheet starting at cell A2.
oSheet.Range("A2").Resize(100, 3).Value = DataArray

'Save the Workbook and quit Excel.
oBook.SaveAs(sSampleFolder & "Book2.xls")
```

```
oSheet = Nothing
oBook = Nothing
oExcel.Quit()
oExcel = Nothing
GC.Collect()
```

If you transfer data by using an array instead of cell by cell, you can realize an enormous performance gain with a lot of data. Consider this line from the earlier code, which transfers data to 300 cells in the worksheet.

```
oSheet.Range("A2").Resize(100, 3).Value = DataArray
```

This line represents two interface requests: one for the **Range** object that the **Range** method returns, and another for the **Range** object that the **Resize** method returns. In contrast, transferring the data cell by cell requires requests for 300 interfaces to **Range** objects. Whenever possible, you can benefit from transferring your data in bulk and reducing the number of interface requests you make.

Use Automation to transfer an ADO recordset to a worksheet range

The object models for Excel 2000 and Excel 2002 provide the **CopyFromRecordset** method for transferring an ADO recordset to a range on a worksheet. The following code illustrates how to automate Excel to transfer the contents of the Orders table in the Northwind sample database by using the **CopyFromRecordset** method.

```
'Create a Recordset from all the records in the Orders table.
Dim sNWind As String
Dim conn As New ADODB.Connection()
Dim rs As ADODB.Recordset
conn.Open("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & _
    sNorthwind & ";")
conn.CursorLocation = ADODB.CursorLocationEnum.adUseClient
rs = conn.Execute("Orders", , ADODB.CommandTypeEnum.adCmdTable)

'Create a new workbook in Excel.
Dim oExcel As Object
Dim oBook As Object
Dim oSheet As Object
oExcel = CreateObject("Excel.Application")
oBook = oExcel.Workbooks.Add
oSheet = oBook.Worksheets(1)

'Transfer the field names to Row 1 of the worksheet:
'Note: CopyFromRecordset copies only the data and not the field
'      names, so you can transfer the fieldnames by traversing the
'      fields collection.
Dim n As Int32
For n = 1 To rs.Fields.Count
    oSheet.Cells(1, n).Value = rs.Fields(n - 1).Name
Next

'Transfer the data to Excel.
oSheet.Range("A2").CopyFromRecordset(rs)

'Save the workbook and quit Excel.
oBook.SaveAs(sSampleFolder & "Book3.xls")
oSheet = Nothing
oBook = Nothing
oExcel.Quit()
oExcel = Nothing
GC.Collect()

'Close the connection
rs.Close()
conn.Close()
```

Note **CopyFromRecordset** works only with ADO **Recordset** objects. A **DataSet** that you create by using ADO.NET cannot be used with the **CopyFromRecordset** method. Several examples in the sections that follow demonstrate how to transfer data to Excel with ADO.NET.

Use Automation to create a QueryTable object on a worksheet

A **QueryTable** object represents a table that is built from data that is returned from an external data source. While you automate Excel, you can create a **QueryTable** by providing a connection string to an OLEDB or an ODBC data source and a SQL string. Excel generates the recordset and inserts the recordset into the worksheet at the location that you specify. Using **QueryTable** objects offers the following advantages over the **CopyFromRecordset** method:

- Excel handles the creation of the recordset and its placement into the worksheet.
- The query can be saved with the **QueryTable** object so that it can be refreshed later to obtain an updated recordset.
- When a new **QueryTable** is added to your worksheet, you can specify that data already existing in cells on the worksheet be shifted to fit the new data (see the **RefreshStyle** property for details).

The following code demonstrates how to automate Excel 2000 or 2002 to create a new **QueryTable** in an Excel worksheet by using data from the Northwind sample database.

```
'Create a new workbook in Excel.
Dim oExcel As Object
Dim oBook As Object
Dim oSheet As Object
oExcel = CreateObject("Excel.Application")
oBook = oExcel.Workbooks.Add
oSheet = oBook.Worksheets(1)

'Create the QueryTable object.
Dim oQryTable As Object
oQryTable = oSheet.QueryTables.Add( _
"OLEDB;Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & _
    sNorthwind & ";", oSheet.Range("A1"), _
    "Select * from Orders")
oQryTable.RefreshStyle = 2 ' xlInsertEntireRows = 2
oQryTable.Refresh(False)

'Save the workbook and quit Excel.
oBook.SaveAs(sSampleFolder & "Book4.xls")
oQryTable = Nothing
oSheet = Nothing
oBook = Nothing
oExcel.Quit()
oExcel = Nothing
```

Use the Clipboard

You can use the Clipboard to transfer data to a worksheet. To paste data into multiple cells on a worksheet, you can copy a string in which columns are delimited by tab characters, and rows are delimited by carriage returns. The following code illustrates how Visual Basic .NET uses the Clipboard to transfer data to Excel.

```
'Copy a string to the Clipboard.
Dim sData As String
sData = "FirstName" & vbTab & "LastName" & vbTab & "Birthdate" & vbCr _
    & "Bill" & vbTab & "Brown" & vbTab & "2/5/85" & vbCr _
    & "Joe" & vbTab & "Thomas" & vbTab & "1/1/91"
System.Windows.Forms.Clipboard.SetDataObject(sData)

'Create a workbook in Excel.
Dim oExcel As Object
Dim oBook As Object
oExcel = CreateObject("Excel.Application")
oBook = oExcel.Workbooks.Add

'Paste the data.
oBook.Worksheets(1).Range("A1").Select()
oBook.Worksheets(1).Paste()

'Save the workbook and quit Excel.
oBook.SaveAs(sSampleFolder & "Book5.xls")
oBook = Nothing
oExcel.Quit()
oExcel = Nothing
GC.Collect()
```

Create a delimited text file that Excel can parse into rows and columns

Excel can open tab-delimited files or comma-delimited files and correctly parse the data into cells. You can use this feature when you want to transfer a lot of data to a worksheet while using little, if any, Automation. This may be a good approach for a client-server program, because the text file can be generated server-side. You can then open the text file at the client, using Automation where it is appropriate.

The following code illustrates how to generate a tab-delimited text file from data that is read with ADO.NET.

```
'Connect to the data source.
Dim objConn As New System.Data.OleDb.OleDbConnection( _
    "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & sNorthwind & ";")
objConn.Open()

'Execute a command to retrieve all records from the Employees table.
Dim objCmd As New System.Data.OleDb.OleDbCommand( _
    "Select * From Employees", objConn)
Dim objReader As System.Data.OleDb.OleDbDataReader
objReader = objCmd.ExecuteReader()

'Read the records in the dataset and write select fields to the
'output file.
FileOpen(1, sSampleFolder & "Book6.txt", OpenMode.Output)
Dim i As Integer, s As String
While objReader.Read()
    'Loop through first 6 fields and concatenate
    'each field, separated by a tab, into s variable.
    s = ""
    For i = 0 To 5
        If Not objReader.IsDBNull(i) Then
            If i = 0 Then 'field 1 is EmployeeId
                s = s & objReader.GetInt32(i).ToString
            ElseIf i = 5 Then 'field 6 is BirthDate
                s = s & objReader.GetDateTime(i)
            Else 'field is a text field
                s = s & objReader.GetString(i)
            End If
        End If
        s = s & Microsoft.VisualBasic.ControlChars.Tab
    Next
    PrintLine(1, s)
End While
FileClose(1)

'Close the reader and the connection.
objReader.Close()
objConn.Close()
```

No Automation was used in the previous code. However, you can use minimal Automation to open the text file and save the file in the Excel workbook format, as follows.

```
'Create a new instance of Excel.
Dim oExcel As Object
oExcel = CreateObject("Excel.Application")

'Open the text file and save it in the Excel workbook format.
oExcel.Workbooks.OpenText(sSampleFolder & "Book6.txt", _
    , , , -4142, , True) 'xlTextQualifierNone=-4142

oExcel.ActiveWorkbook.SaveAs(sSampleFolder & "Book6.xls", _
    -4143) 'xlWorkbookNormal = -4143

'Quit Excel.
oExcel.Quit()
oExcel = Nothing
GC.Collect()
```

Transfer data to a worksheet by using ADO.NET

You can use the Microsoft Jet OLE DB provider to add records to a table in an existing Excel workbook. A "table" in Excel is merely a range of cells; the range may have a defined name. Typically, the first row of the range contains the headers (or field names), and all later rows in the range contain the records.

The following code adds two new records to a table in Book7.xls. The table in this case is Sheet1.

```
'Establish a connection to the data source.
Dim sConnectionString As String
sConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
    "Data Source=" & sSampleFolder & _
    "Book7.xls;Extended Properties=Excel 8.0;"
Dim objConn As New System.Data.OleDb.OleDbConnection(sConnectionString)
```

3. Start Visual Studio .NET. On the **File** menu, click **New** and then click **Project**. Under **Visual Basic Projects**, select **Windows Application**. By default, Form1 is created.
4. Add a reference to the Excel object library. To do this, follow these steps:
   a. On the **Project** menu, click **Add Reference**.
   b. On the **COM** tab, locate **Microsoft Excel 10.0 Object Library**, and then click **Select**.

      Note If you have not already done so, Microsoft recommends that you download and then install the Microsoft Office XP Primary Interop Assemblies (PIAs). For more information about Office XP PIAs, click the following article number to view the article in the Microsoft Knowledge Base:

      328912 (http://support.microsoft.com/kb/328912/) Microsoft Office XP primary interop assemblies (PIAs) are available for download

   c. On the **COM** tab, locate **Microsoft ActiveX Data Objects 2.7 Library**, and then click **Select**.
   d. Click **OK** in the **Add References** dialog box to accept your selections. If you receive a prompt to generate wrappers for the libraries that you selected, click **Yes**.
5. Add a **Combo Box** control and a **Button** control to Form1.
6. Add the following code to Form1.

```
Const sSampleFolder = "C:\ExcelData\"
Const sNorthwind = "C:\Program Files\Microsoft Office\Office10\Samples\Northwind.mdb"

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
  Handles MyBase.Load
    ComboBox1.DropDownStyle = ComboBoxStyle.DropDownList
    Dim aList As String() = _
        {"Use Automation to Transfer Data Cell by Cell ", _
         "Use Automation to Transfer an Array of Data to a Range on a Worksheet ", _
         "Use Automation to Transfer an ADO Recordset to a Worksheet Range ", _
         "Use Automation to Create a QueryTable on a Worksheet", _
         "Use the Clipboard", _
         "Create a Delimited Text File that Excel Can Parse into Rows and Columns", _
         "Transfer Data to a Worksheet Using ADO.NET "}
    ComboBox1.Items.AddRange(aList)
    ComboBox1.SelectedIndex = 0
    Button1.Text = "Go!"
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
  Handles Button1.Click
    Select Case ComboBox1.SelectedIndex
        Case 0 : Automation_CellByCell()
        Case 1 : Automation_UseArray()
        Case 2 : Automation_ADORecordset()
        Case 3 : Automation_QueryTable()
        Case 4 : Use_Clipboard()
        Case 5 : Create_TextFile()
        Case 6 : Use_ADONET()
    End Select
    GC.Collect()
End Sub

Private Function Automation_CellByCell()
    Dim oExcel As Object
    Dim oBook As Object
    Dim oSheet As Object

    'Start a new workbook in Excel.
    oExcel = CreateObject("Excel.Application")
    oBook = oExcel.Workbooks.Add

    'Add data to cells of the first worksheet in the new workbook.
    oSheet = oBook.Worksheets(1)
    oSheet.Range("A1").Value = "Last Name"
    oSheet.Range("B1").Value = "First Name"
    oSheet.Range("A1:B1").Font.Bold = True
    oSheet.Range("A2").Value = "Doe"
    oSheet.Range("B2").Value = "John"
```

```
objConn.Open()

'Add two records to the table.
Dim objCmd As New System.Data.OleDb.OleDbCommand()
objCmd.Connection = objConn
objCmd.CommandText = "Insert into [Sheet1$] (FirstName, LastName)" & _
    " values ('Bill', 'Brown')"
objCmd.ExecuteNonQuery()
objCmd.CommandText = "Insert into [Sheet1$] (FirstName, LastName)" & _
    " values ('Joe', 'Thomas')"
objCmd.ExecuteNonQuery()

'Close the connection.
objConn.Close()
```

When you add records with ADO.NET as shown, the formatting in the workbook is maintained. Each record that is added to a row borrows the format from the row before it. For example, new fields that are added to column B are formatted with right alignment because cell B1 is right-aligned.

Note that when a record is added to a cell or cells in the worksheet, it overwrites any data that those cells previously contained. In other words, rows in the worksheet are not "pushed down" when new records are added. Keep this in mind when you design the layout of data on your worksheets if you plan to the insert new records by using ADO.NET.

For more information about how to use ADO.NET, click the following article numbers to view the articles in the Microsoft Knowledge Base:

301075 (http://support.microsoft.com/kb/301075/) How to connect to a database and run a command by using ADO.NET and Visual Basic .NET

301216 (http://support.microsoft.com/kb/301216/) How to populate a DataSet object from a database by using Visual Basic .NET

301248 (http://support.microsoft.com/kb/301248/) How to update a database from a DataSet object by using Visual Basic .NET

For more information about how to use the Jet OLE DB provider with Excel data sources, click the following article number to view the article in the Microsoft Knowledge Base:

278973 (http://support.microsoft.com/kb/278973/) ExcelADO demonstrates how to use ADO to read and write data in Excel workbooks

257819 (http://support.microsoft.com/kb/257819/) How to use ADO with Excel data from Visual Basic or VBA

Transfer XML data (Excel 2002 only)

Excel 2002 can open any XML file that is well-formed. XML files can be opened directly from the **Open** command on the **File** menu, or programmatically by using either the **Open** or **OpenXML** methods of the **Workbooks** collection. If you create XML files for use in Excel, you can also create style sheets to format the data.

For more information about how to use XML with Excel 2002, click the following article numbers to view the articles in the Microsoft Knowledge Base:

307021 (http://support.microsoft.com/kb/307021/) How to transfer XML data to Microsoft Excel 2002 by using Visual Basic .NET

288215 (http://support.microsoft.com/kb/288215/) Microsoft Excel 2002 and XML

Create the complete sample Visual Basic .NET project

1. Create a new folder to hold the Excel workbooks that the sample will create for you, and then name the folder C:\Exceldata\.
2. Follow these steps to create a new workbook for the sample to write to:
   a. Start a new workbook in Excel.
   b. On Sheet1 of the new workbook, type **FirstName** in cell A1 and **LastName** in cell A2.
   c. Save the workbook as C:\Exceldata\Book7.xls.

```
    'Save the workbook and quit Excel.
    oBook.SaveAs(sSampleFolder & "Book1.xls")
    oSheet = Nothing
    oBook = Nothing
    oExcel.Quit()
    oExcel = Nothing
    GC.Collect()
End Function

Private Function Automation_UseArray()
    Dim oExcel As Object
    Dim oBook As Object
    Dim oSheet As Object

    'Start a new workbook in Excel.
    oExcel = CreateObject("Excel.Application")
    oBook = oExcel.Workbooks.Add

    'Create an array with 3 columns and 100 rows.
    Dim DataArray(99, 2) As Object
    Dim r As Integer
    For r = 0 To 99
        DataArray(r, 0) = "ORD" & Format(r + 1, "0000")
        DataArray(r, 1) = Rnd() * 1000
        DataArray(r, 2) = DataArray(r, 1) * 0.07
    Next

    'Add headers to the worksheet on row 1.
    oSheet = oBook.Worksheets(1)
    oSheet.Range("A1").Value = "Order ID"
    oSheet.Range("B1").Value = "Amount"
    oSheet.Range("C1").Value = "Tax"

    'Transfer the array to the worksheet starting at cell A2.
    oSheet.Range("A2").Resize(100, 3).Value = DataArray

    'Save the workbook and quit Excel.
    oBook.SaveAs(sSampleFolder & "Book2.xls")
    oSheet = Nothing
    oBook = Nothing
    oExcel.Quit()
    oExcel = Nothing
    GC.Collect()
End Function

Private Function Automation_ADORecordset()
    'Create a Recordset from all the records in the Orders table.
    Dim sNWind As String
    Dim conn As New ADODB.Connection()
    Dim rs As ADODB.Recordset
    conn.Open("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & _
        sNorthwind & ";")
    conn.CursorLocation = ADODB.CursorLocationEnum.adUseClient
    rs = conn.Execute("Orders", , ADODB.CommandTypeEnum.adCmdTable)

    'Create a new workbook in Excel.
    Dim oExcel As Object
    Dim oBook As Object
    Dim oSheet As Object
    oExcel = CreateObject("Excel.Application")
    oBook = oExcel.Workbooks.Add
    oSheet = oBook.Worksheets(1)

    'Transfer the field names to Row 1 of the worksheet:
    'Note: CopyFromRecordset copies only the data and not the field
    '      names, so you can transfer the fieldnames by traversing the
    '      fields collection.
    Dim n As Int32
    For n = 1 To rs.Fields.Count
        oSheet.Cells(1, n).Value = rs.Fields(n - 1).Name
    Next
```

```
    'Transfer the data to Excel.
    oSheet.Range("A2").CopyFromRecordset(rs)

    'Save the workbook and quit Excel.
    oBook.SaveAs(sSampleFolder & "Book3.xls")
    oSheet = Nothing
    oBook = Nothing
    oExcel.Quit()
    oExcel = Nothing
    GC.Collect()

    'Close the connection.
    rs.Close()
    conn.Close()
End Function

Private Function Automation_QueryTable()
    'Create a new workbook in Excel.
    Dim oExcel As Object
    Dim oBook As Object
    Dim oSheet As Object
    oExcel = CreateObject("Excel.Application")
    oBook = oExcel.Workbooks.Add
    oSheet = oBook.Worksheets(1)

    'Create the QueryTable object.
    Dim oQryTable As Object
    oQryTable = oSheet.QueryTables.Add( _
    "OLEDB;Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & _
        sNorthwind & ";", oSheet.Range("A1"), _
        "Select * from Orders")
    oQryTable.RefreshStyle = 2 ' xlInsertEntireRows = 2
    oQryTable.Refresh(False)

    'Save the workbook and quit Excel.
    oBook.SaveAs(sSampleFolder & "Book4.xls")
    oQryTable = Nothing
    oSheet = Nothing
    oBook = Nothing
    oExcel.Quit()


    oExcel = Nothing

End Function

Private Function Use_Clipboard()
    'Copy a string to the clipboard.
    Dim sData As String
    sData = "FirstName" & vbTab & "LastName" & vbTab & "Birthdate" & vbCr _
        & "Bill" & vbTab & "Brown" & vbTab & "2/5/85" & vbCr _
        & "Joe" & vbTab & "Thomas" & vbTab & "1/1/91"
    System.Windows.Forms.Clipboard.SetDataObject(sData)

    'Create a new workbook in Excel.
    Dim oExcel As Object
    Dim oBook As Object
    oExcel = CreateObject("Excel.Application")
    oBook = oExcel.Workbooks.Add

    'Paste the data.
    oBook.Worksheets(1).Range("A1").Select()
    oBook.Worksheets(1).Paste()

    'Save the workbook and quit Excel.
    oBook.SaveAs(sSampleFolder & "Book5.xls")
    oBook = Nothing
    oExcel.Quit()
    oExcel = Nothing
    GC.Collect()
End Function
```

```
Private Function Create_TextFile()
    'Connect to the data source.
    Dim objConn As New System.Data.OleDb.OleDbConnection( _
        "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & sNorthwind & ";")
    objConn.Open()

    'Run a command to retrieve all records from the Employees table.
    Dim objCmd As New System.Data.OleDb.OleDbCommand( _
        "Select * From Employees", objConn)
    Dim objReader As System.Data.OleDb.OleDbDataReader
    objReader = objCmd.ExecuteReader()

    'Read the records in the dataset and write select fields to the
    'output file.
    FileOpen(1, sSampleFolder & "Book6.txt", OpenMode.Output)
    Dim i As Integer, s As String
    While objReader.Read()
        'Loop through first 6 fields and concatenate
        'each field, separated by a tab, into s variable.
        s = ""
        For i = 0 To 5
            If Not objReader.IsDBNull(i) Then
                If i = 0 Then 'field 1 is EmployeeId
                    s = s & objReader.GetInt32(i).ToString
                ElseIf i = 5 Then 'field 6 is BirthDate
                    s = s & objReader.GetDateTime(i)
                Else 'field is a text field
                    s = s & objReader.GetString(i)
                End If
            End If
            s = s & Microsoft.VisualBasic.ControlChars.Tab
        Next
        PrintLine(1, s)
    End While
    FileClose(1)

    'Close the reader and the connection.
    objReader.Close()
    objConn.Close()

    'Create a new instance of Excel.
    Dim oExcel As Object
    oExcel = CreateObject("Excel.Application")

    'Open the text file and save it in the Excel workbook format.
    oExcel.Workbooks.OpenText(sSampleFolder & "Book6.txt", _
        , , , -4142, , True) 'xlTextQualifierNone=-4142

    oExcel.ActiveWorkbook.SaveAs(sSampleFolder & "Book6.xls", _
        -4143) 'xlWorkbookNormal = -4143

    'Quit Excel.
    oExcel.Quit()
    oExcel = Nothing
    GC.Collect()
End Function

Private Function Use_ADONET()

    'Verify that the workbook to write to does exist.
    Dim sFile As String = sSampleFolder & "Book7.xls"
    If Dir(sFile) = "" Then
        MsgBox("Please create the workbook Book7.xls and try again.")
        Exit Function
    End If

    'Establish a connection to the data source.
    Dim sConnectionString As String
    sConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=" & sSampleFolder & _
        "Book7.xls;Extended Properties=Excel 8.0;"
    Dim objConn As New System.Data.OleDb.OleDbConnection(sConnectionString)
```

```
        objConn.Open()

        'Add two records to the table named 'MyTable'.
        Dim objCmd As New System.Data.OleDb.OleDbCommand()
        objCmd.Connection = objConn
        objCmd.CommandText = "Insert into [Sheet1$] (FirstName, LastName)" & _
            " values ('Bill', 'Brown')"
        objCmd.ExecuteNonQuery()
        objCmd.CommandText = "Insert into [Sheet1$] (FirstName, LastName)" & _
            " values ('Joe', 'Thomas')"
        objCmd.ExecuteNonQuery()

        'Close the connection.
        objConn.Close()
    End Function
```

Note If you did not install Office to the default folder (C:\Program Files\Microsoft Office), change the **sNorthwind** constant in the code sample to match your installation path for Northwind.mdb.

7. Add the following code to the top of Form1.vb.

```
Imports Microsoft.Office.Interop
```

8. Press F5 to build and then run the sample.

REFERENCES

For more information, visit the following Microsoft Developer Network (MSDN) Web site:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnoxpta/html/vsofficedev.asp **(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnoxpta/html/vsofficedev.asp)**

For more information, click the following article number to view the article in the Microsoft Knowledge Base:

247412 **(http://support.microsoft.com/kb/247412/)** Methods for transferring data to Excel from Visual Basic

APPLIES TO

- Microsoft Excel 2002 Standard Edition
- Microsoft Visual Basic .NET 2002 Standard Edition
- Microsoft .NET Framework 1.1 Service Pack 1

Keywords: kbautomation kbhowtomaster KB306022